CSE220 – Dynamic Single Dimension Arrays

100 points

Topics:

- Arrays
- Dynamic Allocation
- Pointers
- Input & Output

Description

The goal of this assignment is to create arrays during runtime based on user input and then analyze them for interesting information.

You may use either C or C++ for this assignment.

However, make sure you use the correct compiler in your Makefile: g++ for C++ and gcc for C

Use the following Guidelines:

- Give identifiers semantic meaning and make them easy to read (examples numStudents, grossPay, etc).
- Keep identifiers to a reasonably short length.
- User upper case for constants. Use title case (first letter is upper case) for classes. Use lower case with uppercase word separators for all other identifiers (variables, methods, objects).
- Use tabs or spaces to indent code within blocks (code surrounded by braces). This includes classes, methods, and code associated with ifs, switches and loops. Be consistent with the number of spaces or tabs that you use to indent.
- Use white space to make your program more readable.

Important Note:

All submitted assignments must begin with the descriptive comment block. To avoid losing trivial points, make sure this comment header is included in every assignment you submit, and that it is updated accordingly from assignment to assignment.

Programming Assignment:

Instructions:

Based on user input you will create two arrays during runtime via pointers. You will populate the arrays with random numbers. You should limit the random numbers based on user input.

After the arrays are generated you will analyze them with set operations:

- If the set of numbers in both arrays is disjoint, then tell the user and return to the options to build the arrays
- If the set isn't disjoint:
  - Find the union of both arrays and create another array to store the values in the union
  - Find the intersection of both arrays and create another array to store the values in the intersection
  - Find the difference between the first array and the second and create another array to store the values in the difference
  - Find the Symmetric Difference (opposite of the intersection) and create another array to store the values
  - Output each of these set operations

Specifications:

Menu (3% spec)

When the user starts the program, you should welcome them and present them with a simple menu:

```
Welcome to the Set Analyzer!
1 - Manual Input
2 - Random Generation
0 - Exit
```

After finishing an option 1 or 2 you should return to the main menu.

Menu Option 1 (5% spec)

Ask the user for two array sizes.

Loop through the arrays in turn asking the user for values to fill the arrays.

Then perform the set operation analysis on the arrays.

Menu Option 2 (5% spec)

Ask the user for two array sizes.

Ask the user for a minimum and maximum integer.

Call on the Generate Array function for each array.

Perform the set operation analysis on the arrays.

Return to Main Menu (2% Spec)

After finishing Menu Option 1 or 2 return to the Main Menu.

Required Functions (75% Spec)

Write a function for each of the Set Operations:

- Generate array
  - Takes a size, a minimum, and a maximum
  - Dynamically allocates an array
  - Fills it with random numbers between minimum & maximum
  - Returns the array
- Disjoint
  - Takes two integer arrays and two sizes
  - Returns Boolean/Integer if they are disjoint
- Intersection
  - Takes two integer arrays and two sizes
  - Returns a new array
- Union
  - Takes two integer arrays and two sizes
  - Returns new array that contains all elements of both
- Left Difference
  - Takes two integer arrays and two sizes
  - Returns a new array that is what is left in first parameter after removing the intersection with the second parameter
- Symmetric Difference
  - Takes two integer arrays and two sizes
  - Returns a new array that is what is left in both the first and second array after the intersection is removed from both
- Output Array
  - Takes an array of integers and its size
  - Outputs the array
- Analysis function
  - Takes two integer arrays and two sizes
  - Performs all of the above tasks
  - Calls output Array on the results

Each set operation will need to dynamically allocate a new array and return it. Remember this is done through pointers, and remember that you can return a pointer from a function.

Makefile (10% Spec)

Create a Makefile to compile your code. The grader should be able to simply type "make" into the terminal and get your executable.

Also include a target for "make clean" which should remove any .o files, the executable file and any backup files made by emacs for vim.

Sample Output:

```
Welcome to the Set Analyzer!
1 - Manual Input
2 - Random Generation
0 - Exit
>>
```

If user selects Menu Option 1

```
You have chosen Manual Input
Enter Array Size 1: <user input value>
Enter Array Size 2: <user input value>

Now collecting input for Array #1:
Enter values for each index:
0: <value>
1: <value>
…


Now collecting input for Array #2:
Enter values for each index:
0: <value>
1: <value>
…
```

If user selects Menu Option 2

```
You have chosen Random Generation

Enter Array Size 1: <user input value>
Enter Array Size 2: <user input value>

Enter minimum integer value: <user input>
Enter maximum integer value: <user input>
```

After building the array, do the analysis:

```
Running set operations!
Disjoint? <yes/no>
```

If they are disjoint, return the main menu, if they are not continue on:

```
Union: [ <values separated by commas> ]
Intersection: [ <values separated by commas> ]
Left Difference: [ <values separated by commas> ]
Symmetric Difference: [ <values separated by commas> ]
```

Return to the main menu

Sample Run

```
Welcome to the Set Analyzer!
1 - Manual Input
2 - Random Generation
0 - Exit
>> 1

You have chosen Manual Input
Enter Array Size 1: 5
Enter Array Size 2: 5

Now collecting input for Array #1:
Enter values for each index:
0: 1
1: 2
2: 3
3: 4
4: 5

Now collecting input for Array #2:
Enter values for each index:
0: 4
1: 5
2: 6
3: 7
4: 8

Running set operations!
Disjoint? NO
Union: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
Intersection: [ 4, 5 ]
Left Difference: [ 1, 2, 3 ]
Symmetric Difference: [ 1, 2, 3, 6, 7, 8]
```

Return to the main menu

Grading of Programming Assignment

The TA will grade your program following these steps:

(1) Compile the code. If it does not compile a U or F will be given in the Specifications section. This will probably also affect the Efficiency/Stability section.

(2) The TA will read your program and give points based on the points allocated to each component, the readability of your code (organization of the code and comments), logic, inclusion of the required functions, and correctness of the implementations of each function.

Rubric:

	Levels of Achievement						
Criteria	A	B	C	D	E	U	F
Specifications Weight 50.00%	100 % The program works and meets all of the specifications.	85 % The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	75 % The program produces mostly correct results but does not display them correctly and/or missing some specifications	65 % The program produces partially correct results, display problems and/or missing specifications	35 % Program compiles and runs and attempts specifications, but several problems exist	20 % Code does not compile and run. Produces excessive incorrect results	0 % Code does not compile. Barely an attempt was made at specifications.
Code Quality Weight 20.00%	100 % Code is written clearly	85 % Code readability is less	75 % The code is readable only by someone who knows what it is supposed to be doing.	65 % Code is using single letter variables, poorly organized	35 % The code is poorly organized and very difficult to read.	20 % Code uses excessive single letter identifiers. Excessively poorly organized.	0 % Code is incomprehensible
Documentation Weight 15.00%	100 % Code is very well commented	85 % Commenting is simple but solid	75 % Commenting is severely lacking	65 % Bare minimum commenting	35 % Comments are poor	20 % Only the header comment exists identifying the student.	0 % Non existent
Efficiency Weight 15.00%	100 % The code is extremely efficient without sacrificing readability and understanding.	85 % The code is fairly efficient without sacrificing readability and understanding.	75 % The code is brute force but concise.	65 % The code is brute force and unnecessarily long.	35 % The code is huge and appears to be patched together.	20 % The code has created very poor runtimes for much simpler faster algorithms.	0 % Code is incomprehensible

What to Submit?

You are required to submit your solutions in a compressed format (.zip). Zip all files into a single zip file. Make sure your compressed file is labeled correctly - <lastname>_<firstname>_assn3.c OR .cpp

The compressed file MUST contain the following:

- <lastname>_<firstname>_assn3.c OR .cpp
- Makefile

No other files should be in the compressed folder.

If multiple submissions are made, the most recent submission will be graded, even if the assignment is submitted late.

Where to Submit?

All submissions must be electronically submitted to the respected homework link in the course web page where you downloaded the assignment.

Academic Integrity and Honor Code.

You are encouraged to cooperate in study group on learning the course materials. However, you may not cooperate on preparing the individual assignments. Anything that you turn in must be your own work: You must write up your own solution with your own understanding. If you use an idea that is found in a book or from other sources, or that was developed by someone else or jointly with some group, make sure you acknowledge the source and/or the names of the persons in the write-up for each problem. When you help your peers, you should never show your work to them. All assignment questions must be asked in the course discussion board. Asking assignment questions or making your assignment available in the public websites before the assignment due will be considered cheating.

*The instructor and the TA will **CAREFULLY** check any possible proliferation or plagiarism. We will use the document/program comparison tools like MOSS (Measure Of Software Similarity: http://moss.stanford.edu/) to check any assignment that you submitted for grading. The Ira A. Fulton Schools of Engineering expect all students to adhere to ASU's policy on Academic Dishonesty. These policies can be found in the Code of Student Conduct:*

http://www.asu.edu/studentaffairs/studentlife/judicial/academic_integrity.htm

ALL cases of cheating or plagiarism will be handed to the Dean's office. Penalties include a failing grade in the class, a note on your official transcript that shows you were punished for cheating, suspension, expulsion and revocation of already awarded degrees.